HTML5を使った
Java Script
[完全入門]

河西朝雄著

HTML5＋CSS＋JavaScriptを組み合わせた
Webアプリを作って学生企業家をめざせ！

JavaScript

各節ごとにそこで扱うテーマを本文で解説しています。
「その解説を読んで」、「それに関連した例題を試し」、「練習問題で考える」という
３ステップの基本動作を繰り返すことで学習効果が上がります。

KASAI.SOFTWARELAB

定価1,500円＋税

はじめに

JavaScript は HTML 内に記述し、Web ページを動的に制御するためのスクリプト言語です。JavaScript から HTML と CSS をプログラムで制御することによりパソコン、スマートフォン、タブレットで動作する動的な Web アプリを簡単に作ることができます。

本書は HTML5＋CSS＋JavaScript を組み合わせた Web アプリを作るための JavaScript の入門書です。プログラム経験のない人が簡単に JavaScript でプログラムを作成できるように、基礎的な技術内容を系統的に解説するもので、以下のような特徴があります。

●ブラウザとテキストエディタさえあれば簡単にプログラムできます

JavaScript はブラウザに内蔵されているので、特別なソフトが必要でなくインストールなどの作業が必要ないので、初心者には敷居が低いと思います。VisualC++や Java などをダウンロードしインストールする場合、初心者は 1 回でできず苦戦したり、あきらめてしまう場合が多いようです。

●C 言語で学ぶより学習効果が得られます

学校の授業などでプログラムの入門として C 言語を学ぶ例が多いでようです。C の言語仕様にはグラフィックス処理は含まれておらず､各 C 処理系でのサポートとなりますので、統一性がありません。

JavaScript を使えば、HTML5 から使用できるようになった<canvas>要素（タグ）へのグラフィックス処理を簡単に行うことができます。1 章でグラフィックスを用いて JavaScript の基本的な文法を説明します。初心者にとってグラフィックス処理で得られる結果は視覚的に興味がわく題材となります。

JavaScript の基本言語仕様は C 言語をベースにしているので、C 言語の文法を学んだのと同等の効果が得られます。

●Web アプリの基礎が学べます

本書は「JavaScript 完全入門」と題していますが、JavaScript の文法だけを学習することが目的ではありません。HTML5+CSS+JavaScript による Web アプリを作る上での基礎的な技術内容として「2 章　イベント処理とオブジェクト操作」、「3 章　UI 要素の操作」、「4 章　マウス操作」、「5 章　一定時間ごとの処理」、「6 章　JavaScript の標準オブジェクト」、「7 章　マルチメディア」、「8 章　スマートフォン・タブレット専用機能」を解説します。さらにこれらの基礎技術を応用して「9 章　ラケットゲーム」、「11 章　リバーシーゲ

ーム」で示すような実用的な Web アプリが作れるようにすることを目的にしています。

●データ構造やアルゴリズムについて学べます

JavaScript の言語仕様や基礎技術を学んだだけでは高度なプログラムを作ることはできません。データ構造やアルゴリズムといったプログラミングの概念を「10 章　データ構造とアルゴリズム」で解説します。

●パソコン、スマートフォン、タブレットで動作します

本書のプログラムはパソコン上で作成していますが、スマートフォンやタブレット上でも動作します。スマートフォンやタブレットで動作できればより身近なものに感じられはずです。8 章 8-1 の「「補足」スマートフォンやタブレットで動作確認する方法」を参照してください。

●授業や講習会などで使い易いように例題と練習問題がペアであります

各節ごとにそこで扱うテーマを本文で解説しています。「その解説を読んで」、「それに関連した例題を試し」、「練習問題で考える」という 3 ステップの基本動作を繰り返すことで学習効果が上がります。応用サンプルではさらに発展的な内容を学習できます。

●Appendix に HTML、CSS、JavaScript に関する言語仕様をまとめてあります

本書で扱う HTML と CSS は通常の Web ページを作るのに必要な知識ではなく、JavaScript から制御されるオブジェクトという観点で扱います。従って本書では主に JavaScript の説明が主で HTML と CSS は JavaScript を使ってプログラムする上で必要な最小限の内容に絞って解説しています。HTML と CSS に関する簡単な解説を Appendix1 にまとめました。JavaScript の詳細な言語仕様の解説を Appendix2 にまとめました。

JavaScript の文法以外にも学ばなければならないことがたくさんあり、高度な内容も含まれていますが解説はできるだけ分り易くしたつもりです。本書を読んだ方が一人でも多く、プログラムって面白いな、色々自分でも作ってみたいなと思っていただけたら幸いです。

2015 年 4 月

河西　朝雄

目次

1章　グラフィックスを用いたJavaScript入門

HTML5から使用できるようになった<canvas>要素（タグ）へのグラフィックス処理を例にしてJavaScriptの基本的な文法を説明します。初心者にとってグラフィックス処理で得られる結果は視覚的に興味がわく題材となります。

プログラムを作る上で基本となることは、プログラムの流れを制御する流れ制御文、データをまとめて管理する配列、一連の処理内容をひとかたまりにまとめて記述しそれを呼び出して使う関数などです。このようなプログラムを作る際の決まりを定めたものを言語仕様といいます。この章では以下のような基本的な言語仕様について説明します。

・変数と演算子
・for文
・if else文
・二重ループ
・1次元配列
・2次元配列
・関数
・その他の制御文
　while文、else if文、switch case文、連想配列とfor in文

1-1　JavaScript の概要

1.　JavaScript とは

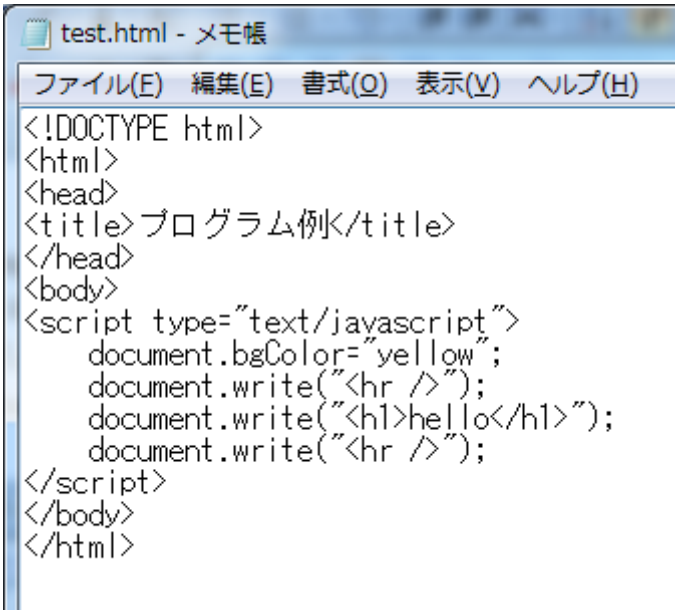
JavaScript は HTML 内に記述し、Web ページを動的に制御するためのスクリプト言語です。JavaScript の特徴は以下の通りです。

・JavaScript でプログラムを作る上で必用なソフトはメモ帳などのエディタと Internet Explorer や Chrome などのブラウザだけで良い。
・HTML の<script>ブロック内に JavaScript プログラムを記述する。
・プログラムの書式や言語仕様は C 言語と極めて近い。
・Java という名称がついているが、オブジェクトの扱いは Java とは全く異なる。
・JavaScript で扱えるオブジェクトを大別すると、JavaScript で規定している標準オブジェクト（コアオブジェクト)、W3C で規定している HTML や XML で記述された各要素を取り扱うための DOM オブジェクト、明確な規定はないが多くのブラウザで採用されているブラウザ関連の Browser オブジェクトの 3 種類となる。
・HTML タグ中の onXXX 属性に JavaScript 関数を指定してマウス操作、フォーム要素のチェックなどの各種イベント処理を行なうことができる。addEventListener メソッドを使ってイベントリスナーを追加することもできる。
・Style オブジェクトを使ってオブジェクトの位置やサイズを制御できる。
・キャンバスオブジェクトに対しグラフィックス処理ができる。
・ビデオ、オーディオ、マップなどのオブジェクトを操作できる。
・マウスのドラッグ＆ドロップ処理ができる。
・スマートフォンやタブレットでのタッチ処理やマルチタッチ処理ができる。
・スマートフォンやタブレットに内蔵されている Geolocation(ジオロケーション)やセンサーを操作できる。

「注」スクリプト言語
スクリプト言語とはメインになる言語の中に埋め込んで、機能を補完したり拡張したりするための言語です。

2. JavaScript の書式

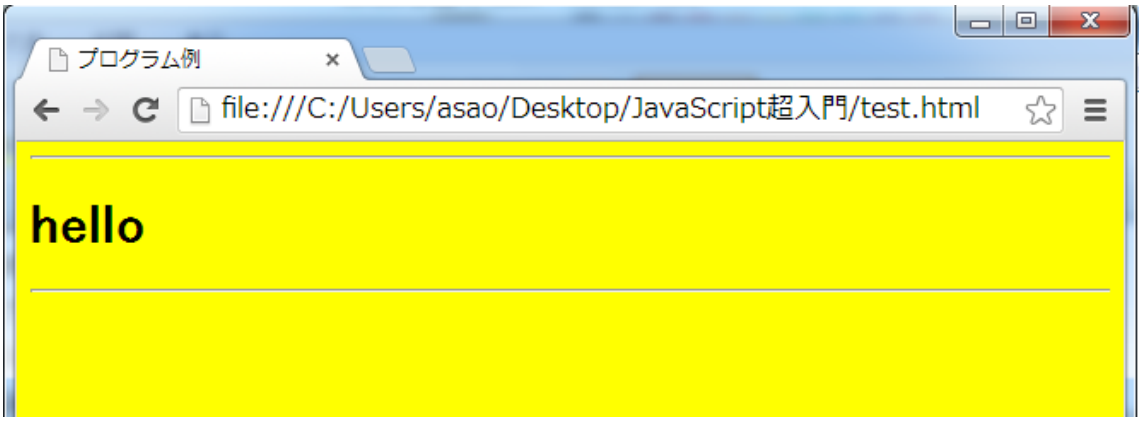
JavaScript は HTML ファイル内に<script>・・・</script>タグで囲んだスクリプトブロック内に記述します。JavaScript で水平線と「hello」という文字をブラウザに表示するプログラムを「test.html」としてメモ帳で編集した例を以下に示します。



type 属性に"text/javascript"を指定することで、このスクリプトブロックは JavaScript で記述されていることを示します。"text/javascript"は省略可能です。スクリプトブロックは<head>部または<body>部に置くことができます。

この JavaScript プログラムにおいて document はこのスクリプトブロックが置かれている HTML ファイルを表すオブジェクトです。bgColor は document オブジェクトの背景色を示すプロパティです。write は（）内の内容を document オブジェクトに出力するメソッドです。（）内にはダブルクオーテーションで囲んだ"<h1>hello</h1>"のような HTML 文を書きます。JavaScript の文の終わりにはセミコロン（；）を置きます。

このプログラム「test.html」をブラウザで見ると次のようになります。



3. オブジェクト、メソッド、プロパティ

オブジェクトとは処理を行う対象物を表し、オブジェクトを操作するための命令としてメソッドとプロパティがあります。使用できるメソッドとプロパティはオブジェクトごとに決められています。先の例では document というオブジェクトに対し write メソッドや bgColor プロパティを使用しています。

メソッド、プロパティの一般的な書式は以下のようになります。オブジェクトとメソッドまたはプロパティの間は「.」で区切ります。

```
オブジェクト.メソッド(引数 1,引数 2,・・・);
オブジェクト.プロパティ[＝値] ;
```

メソッドは与えられた複数の引数（ない場合もある）をオブジェクトに渡し、そこである処理をした後、戻り値があればそれを返します。

プロパティはオブジェクトの属性を示す変数のようなもので、１つの値を取得あるいは設定します。「オブジェクト.プロパティ＝値」なら値の設定となり、「オブジェクト.プロパティ」なら値の取得になります。

1-2　グラフィックス処理の概要

HTML5 の<canvas>要素（タグ）を利用すれば、JavaScript から lineTo や strokeRect などのメソッドを使って、直線、矩形（四角）、円などの 2 次元グラフィックス描画を行うことができます。

1.　<canvas>要素（タグ）

HTML5 ではグラフィックス描画を行うための領域を<canvas>タグで指定することができます。width と height 属性にキャンバスの幅と高さをピクセル単位で指定します。

```
<canvas id="canvas" width="400" height="400"></canvas>
```

「注」ピクセル単位を明示するには"400px"とします。

```
<canvas id="canvas" width="400px" height="400px"></canvas>
```

2.　<canvas>領域への描画手順

<canvas>タグで指定した領域にグラフィックス描画を行うにはキャンバスオブジェクトを取得し、さらにそのキャンバスオブジェクトから実際にグラフィックス描画を行うためのコンテキストオブジェクトを取得します。getContext に指定できる引数は現在「2d:2 次元グラフィックス」だけです。このコンテキストオブジェクトに対し strokeRect メソッドや fillRect メソッドを使って図形を描画します。

```
var canvas = document.getElementById("canvas");
if(canvas.getContext){
    var context = canvas.getContext("2d");
    // context に対し描画メソッドを適用する
}
```

3.　描画色

描画を行う際の描画色は strokeStyle プロパティに設定します。指定する色は"blue"のような色名、"#0000ff"のような 16 進数の RGB 値、"rgb(0,0,255)"のような rgb 関数が指定できます。

```
context.strokeStyle = "blue"; //  描画色
```

矩形や円の内部を塗りつぶす色は fillStyle プロパティに設定します。

context.fillStyle = "blue"; // 塗りつぶす色

4. 矩形の描画

矩形の描画は strokeRect メソッドを使います。矩形の内部を塗りつぶすには fillRect メソッドを使います。

矩形の描画メソッド	機能
context.strokeRect(x,y,w,h)	左上隅座標を(x,y)、幅を w、高さを h とする矩形を描く。
context.fillRect(x,y,w,h)	左上隅座標を(x,y)、幅を w、高さを h とする矩形内部を塗りつぶす。ただし外枠は描画されない。
context.clearRect(x,y,w,h)	左上隅座標を(x,y)、幅を w、高さを h とする矩形内部を背景色（デフォルトで白）でクリア。

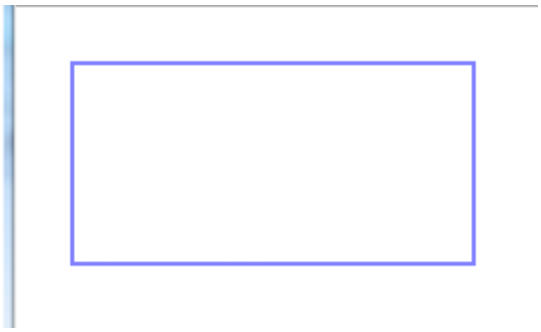
「注」領域のクリア

「context.clearRect(x,y,w,h)」を行うと指定した四角の領域が背景色（デフォルトで白）でクリアされます。ブラウザによっては beginPath でパスに設定しなかった図形（イメージなど）は clearRect で消去できない場合があります。この場合は白で fillRect します。

```
context.fillStyle="white";
context.fillRect(x,y,w,h);
```

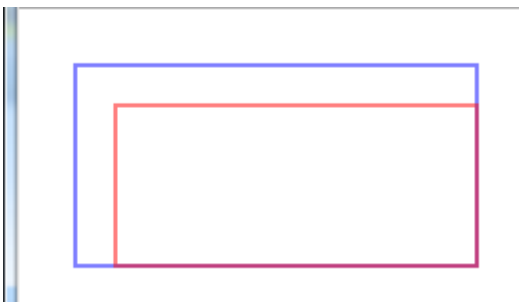
「例題 1-2-1」左上隅座標を（20,20)、幅 200、高さ 100 の矩形を青で描きます。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="canvas" width="400" height="400"></canvas>
<script type="text/javascript">
    var canvas = document.getElementById("canvas");
    if(canvas.getContext){
        var context = canvas.getContext("2d");
        context.strokeStyle = "blue"; // 描画色
        context.strokeRect(20, 20, 200, 100);
    }
</script>
</body>
</html>
```



「練習問題 1-2-1」赤の四角を追加しなさい。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="canvas" width="400" height="400"></canvas>
<script type="text/javascript">
	var canvas = document.getElementById("canvas");
	if(canvas.getContext){
		var context = canvas.getContext("2d");
		context.strokeStyle = "blue"; //  青
		context.strokeRect(20, 20, 200, 100);

		context.strokeStyle =    ①    ; //  赤
		                ②              ;
	}
</script>
</body>
</html>
```



5. 直線の描画

矩形を描く strokeRect メソッドは、このメソッドでキャンバスに直接描画を行いました。他の図形（直線や円など）はパスに対する描画メソッドを使って一旦パスに対し描画を行い、その後 stroke メソッドを使ってパス情報をキャンバスに描画します。パスとは直線、円（円弧）、ベジェ曲線などの図形の各点の情報を繋いだ経路です。

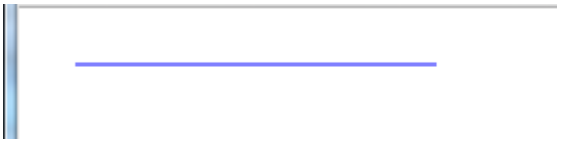
パスへの描画を行うにはまず、beginPath メソッドを使って現在のパスをリセット（クリア）してから行います。

パスに対し直線を描くためのメソッドとして以下があります。

パスに対し直線を描くメソッド	機能
context.moveTo(x,y)	描画現在位置を(x,y)に移動。
context.lineTo(x,y)	描画現在位置から(x,y)に直線を描く。

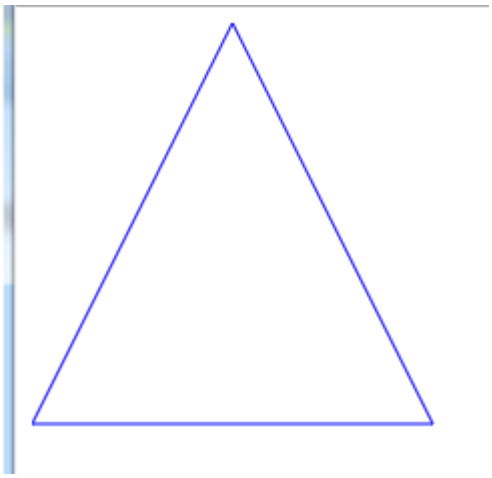
「例題 1-2-2」(20,20)－(200,20)間に青色の直線を描きます。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="canvas" width="400" height="400"></canvas>
<script type="text/javascript">
    var canvas = document.getElementById("canvas");
    if(canvas.getContext){
        var context = canvas.getContext("2d");
        context.strokeStyle = "blue"; //  描画色
        context.beginPath();          //  パスの開始
        context.moveTo(20, 20);
        context.lineTo(200, 20);
        context.stroke();             //  パスの描画
    }
</script>
</body>
</html>
```



「練習問題 1-2-2」三角形を描きなさい。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="canvas" width="400" height="400"></canvas>
<script type="text/javascript">
	var canvas = document.getElementById("canvas");
	if(canvas.getContext){
		var context = canvas.getContext("2d");
		context.strokeStyle = "blue"; //  描画色
		context.beginPath();		//  パスの開始
		context.moveTo(0, 200);
		context.lineTo(200, 200);
		　　①　　;
		　　②　　;
		context.stroke();		//  パスの描画
	}
</script>
</body>
</html>
```



6. 円と円弧

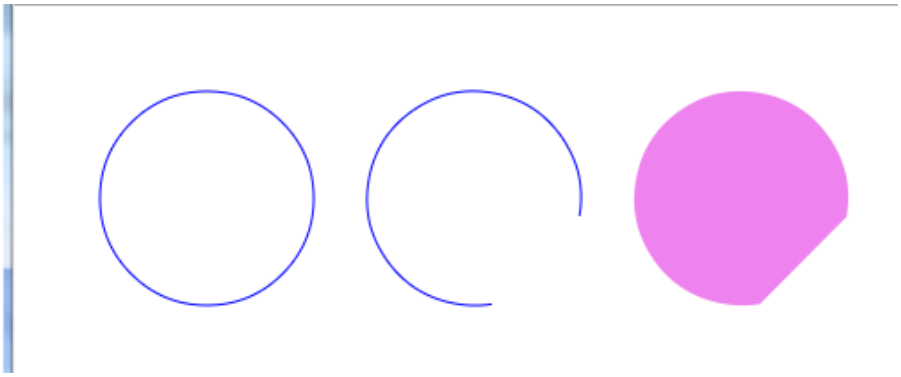
円と円弧は直線の場合と同様に arc メソッドで一度パスに描画してから、stroke あるいは fill メソッドでキャンバスに描画します。

円と円弧の描画メソッド	機能
context.arc(x,y,r,start,end,clockwise)	中心(x,y)、半径 r の円（円弧）を角度 start～end（ラジアン）まで描きます。clockwise に true を指定すると反時計回り、false を指定すると時計回りで円弧を描きます。

「例題 1-2-3」円、円弧、円弧の内部を塗った図形を描きます。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="canvas" width="600" height="400"></canvas>
<script type="text/javascript">
    var canvas = document.getElementById("canvas");
    if(canvas.getContext){
        var context = canvas.getContext("2d");
        context.strokeStyle = "blue"; //  輪郭の色
        context.fillStyle = "violet"; //  塗る色

        context.beginPath();
        context.arc(100, 100, 60, 0, Math.PI*2, false);
        context.stroke();

        context.beginPath();
        context.arc(250, 100, 60, 10 * Math.PI / 180, 80 * Math.PI / 180, true);
        context.stroke();

        context.beginPath();
        context.arc(400, 100, 60, 10 * Math.PI / 180, 80 * Math.PI / 180, true);
```

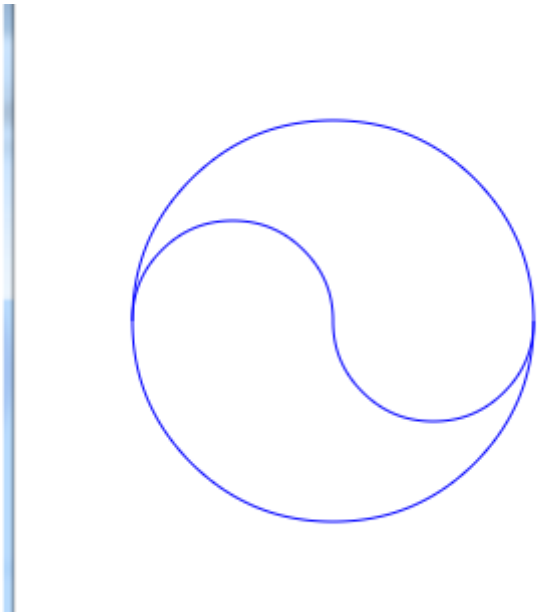
```
            context.fill();
        }
</script>
</body>
</html>
```



「練習問題 1-2-3」 半径 100 の円を 1 つ描き、その中に半径 50 の半円を 2 つ描きなさい。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="canvas" width="600" height="400"></canvas>
<script type="text/javascript">
        var canvas = document.getElementById("canvas");
        if(canvas.getContext){
            var context = canvas.getContext("2d");
            context.strokeStyle = "blue"; //  輪郭の色

            context.beginPath();
            context.arc(150, 150, 100, 0, Math.PI*2, false);
            context.stroke();

            context.beginPath();
            ____________________①____________________;
            context.stroke();

            context.beginPath();
```

```
                    ②                    ;
        context.stroke();
    }
</script>
</body>
</html>
```



7. テキストの描画

テキストの描画は strokeText メソッドまたは fillText メソッドで行います。

テキストの描画メソッド	機能
context.strokeText(テキスト,x, y)	輪郭で描画します。
context.fillText(テキスト,x, y)	塗りつぶしで描画します。

描画位置の(x,y)はテキストの左下隅の座標です。font プロパティにフォントサイズ、書体（italic/bold）、フォント名などを空白で区切って指定します。フォント名に空白が含まれる場合はフォント名を(')で囲みます。

```
context.font = "36px 'ＭＳ　Ｐゴシック'";
```

「例題 1-2-4」(10,40)位置に「ストロークテキスト」というテキストを青色でフォントサイズ 32px、フォント名「ＭＳ　Ｐゴシック」のフォントで描きます。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="canvas" width="400" height="400"></canvas>
<script type="text/javascript">
	var canvas = document.getElementById("canvas");
	if(canvas.getContext){
		var context = canvas.getContext("2d");
		context.strokeStyle = "blue";
		context.font = "32px 'ＭＳ　Ｐゴシック'";
		context.strokeText("ストロークテキスト", 10, 40);
	}
</script>
</body>
</html>
```

「練習問題 1-2-4」3 人の武将の名前を描きなさい。

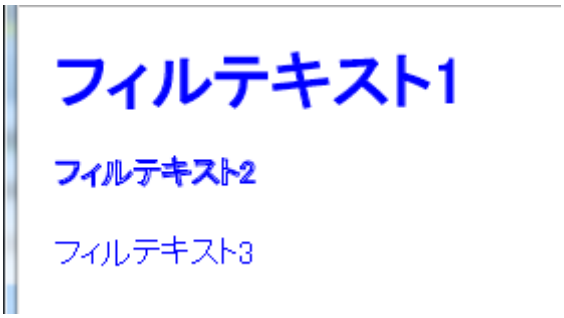
```
<!DOCTYPE html>
<html>
<body>
<canvas id="canvas" width="400" height="400"></canvas>
<script type="text/javascript">
	var canvas = document.getElementById("canvas");
	if(canvas.getContext){
		var context = canvas.getContext("2d");
		context.strokeStyle = "blue";
		context.font = "32px 'ＭＳ　Ｐゴシック'";

		context.strokeText("織田信長", 10, 40);
		                  ①                  ;
```

```
        context.strokeText("徳川家康", 10, 120);
    }
</script>
</body>
</html>
```

「補足」fillText でテキストを描く場合、フォントサイズの大きいものは strokeText と fillText を併用した方がきれいに描けますが、フォントサイズが小さいものは fillText のみの方がきれいに描けます。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="canvas" width="400" height="400"></canvas>
<script type="text/javascript">
    var canvas = document.getElementById("canvas");
    if(canvas.getContext){
        var context = canvas.getContext("2d");
        context.strokeStyle = "blue";
        context.fillStyle = "blue";
        context.font = "32px 'ＭＳ　Ｐゴシック'";
        context.strokeText("フィルテキスト 1", 10, 40);
        context.fillText("フィルテキスト 1", 10, 40);

        context.font = "16px 'ＭＳ　Ｐゴシック'";
        context.strokeText("フィルテキスト 2", 10, 80);
        context.fillText("フィルテキスト 2", 10, 80);
```

```
            context.fillText("フィルテキスト 3", 10, 120);
        }
</script>
</body>
</html>
```



1-3　変数と演算子

プログラムを書く上で、変数の役割をしっかり押さえておく必要があります。変数はプログラムの進行によってその値が変わります。この意味から「変わる数＝変数」と呼びます。これに対し 10 や"hello"などの変化しないものを定数と呼びます。定数は数値定数と文字列定数に分かれます。

1.　変数の宣言

変数は使用する前に、予約語の var を使って宣言します。var は variable を意味します。以下は変数名が「i」の変数を宣言しています。

```
var i;
```

複数の変数を宣言するには変数をコンマ（,）で区切ります。

```
var i,j;
```

それぞれ単独に

```
var i;
var j;
```

と宣言しても良いです。

JavaScript では変数は宣言しなくても使用できますが、プログラムの見やすさ、安全性を考えると宣言すべきです。

2.　変数の初期化

変数を宣言したままだと、その変数の内容は不定で、0 になっている保証はありません。そこで、変数は初期化を行う必要があります。

変数 s の宣言時に 0 で初期化するには次のようにします。

```
var s=0;
```

変数の宣言と代入を分けて、次のようにすることもできます。

```
var s;
s=0;
```

プログラム中で一度だけ初期化を行えば良いのであれば前者を、複数の場所でそのつど初期化が必要な場合には後者を使います。

複数の変数を宣言時に初期化するには次のようにします。

```
var sum=0,ave=0;
```

3. 変数名の付け方

変数に付ける名前を変数名と呼びます。変数名の名前付けのおおよその規則は以下のようなものです。

・英字で始まる英数字。a1 や sum など。
・英大文字と英小文字を区別する。SUM,Sum,sum はいずれも異なる変数名となる。
・予約語を使用してはいけないが、それを含むことはできる。たとえば、for は認められないが force は認められる。

変数名はユーザが決めれば良いわけですが、ある程度の共通ルールに従った方がプログラムを読みやすくします。

・for などのループ変数には i,j,k などを使う
・変数の役割を連想できる変数名を使う。合計なら sum など。

4. 変数への代入と更新

変数へのデータの代入を行う場合、左辺には 1 つの変数しか指定できませんが、右辺には、定数、変数、これらを演算子で結んだ式を指定することができます。

```
var a,b,c,x=10,y=20;
a=1;
b=x;
c=x+y+30;
```

これで、a に 1、b に 10、c に 60 が入ります。
プログラムでは次のような表現を良く使います。

```
sum=sum+10;
```

これを数学の等式と考えるとおかしなことになりますが、プログラムでは、右辺の sum+10 を行なった結果の値を左辺の変数 sum に代入するという意味になります。

従って、以下を行なえば、sum の値は最初 1 ですが、sum+10 で 11 となり、その値が sum に代入されるので、sum の値は 11 になります。

```
sum=1;
sum=sum+10;
```

「○＝○＋□」のように左辺と右辺に同じ変数○がある場合は変数の内容に□を足した値が新しい変数の値になります。このように変数の内容に値を足すなどして新しい値にすることを、変数の更新といいます。

```
sum=sum+10;
```

は

```
sum+=10;
```

と書くことができ、+=を複合代入演算子といいます。本書では後者の使い方をします。+=の他に-=、*=、/=などが使えます。

5. 算術演算子

変数 sum にいくつかのデータの合計を求めるには、「sum=103+1020+・・・+5020;」のように書いておけば、あとはコンピュータが計算結果を求めて、変数 sum に代入してくれます。ここで、足し算をしている「+」を演算子と呼びます。計算の基本は、「足す」、「引く」、「掛ける」、「割る」の 4 種類で、これを行なう演算子を算術演算子（加減乗除演算子）と呼びます。この他に余りを求める演算子と負符号を示す演算子があります。これらの演算子は次の記号を使います。

演算子	意味
+	加算
−	減算
*	乗算
/	除算
%	余り（剰余）
−	負符号

数学の記号では加算の「＋」と減算の「－」は同じですが、乗算の「×」が「*」、除算

の「÷」が「/」となります。

演算処理を行なう際には演算子の優先順位があります。

sum=10+20*30;

という計算式があったとき、「*」の優先順位は「+」より高いと決めているので、「20*30」が先に計算され、次にその結果と 10 が加算され、最後に「=」演算子で sum に代入されます。「=」演算子は一番優先順位が低いのです。これはうまいルールなのです。もし「=」の優先順位が他の演算子より高かったら「sum=10」が先に行なわれてしまいます。こうしたルールを作っておくことで、人間の常識とある程度一致した記述ができるのです。

ところで、どうしても優先順位を変えたければ、() を使って

sum=(10+20)*30;

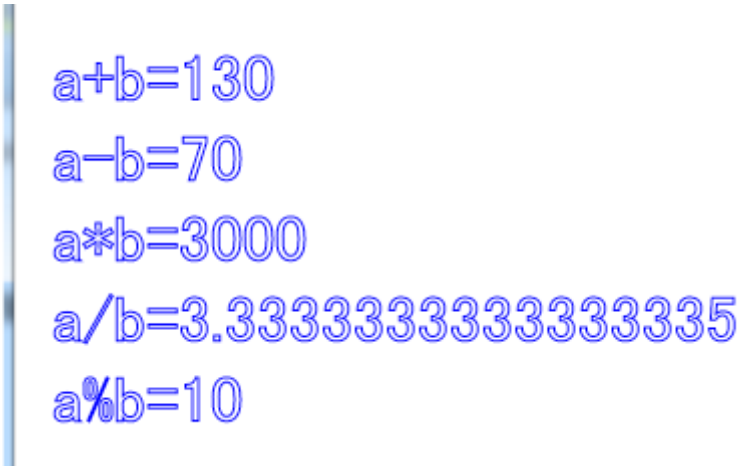
とします。これで 10+20 が先に計算されます。以下のような同じ優先順位なら、左から演算されます。

a+b+c

「例題 1-3」変数 a と b の加算、減算、乗算、除算、剰余の結果を表示します。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="canvas" width="400" height="400"></canvas>
<script type="text/javascript">
    var canvas = document.getElementById("canvas");
    if(canvas.getContext){
        var context = canvas.getContext("2d");
        context.strokeStyle = "blue";
        context.font = "32px 'ＭＳ　Ｐゴシック'";
        var a=100,b=30,tasu,hiku,kakeru,waru,amari;
        tasu=a+b;
        hiku=a-b;
        kakeru=a*b;
        waru=a/b;
        amari=a%b;
```

```
            context.strokeText("a+b="+tasu, 10, 40);
            context.strokeText("a-b="+hiku, 10, 80);
            context.strokeText("a*b="+kakeru, 10, 120);
            context.strokeText("a/b="+waru, 10, 160);
            context.strokeText("a%b="+amari, 10, 200);
        }
</script>
</body>
</html>
```



「練習問題 1-3」変数の tasu,hiku,kakeru,waru,amari を使わずに strokeText メソッドの引数に直接、演算式を指定しなさい。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="canvas" width="400" height="400"></canvas>
<script type="text/javascript">
        var canvas = document.getElementById("canvas");
        if(canvas.getContext){
            var context = canvas.getContext("2d");
            context.strokeStyle = "blue";
            context.font = "32px 'ＭＳ　Ｐゴシック'";
            var a=100,b=30;
            context.strokeText("a+b="+(a+b), 10, 40);
```

```
            context.strokeText("a-b="+(a-b), 10, 80);
            context.strokeText("a*b="+(a*b), 10, 120);
            context.strokeText("a/b="+  ①  , 10, 160);
            context.strokeText("a%b="+  ②  , 10, 200);
        }
</script>
</body>
</html>
```

1-4　for 文

プログラムの流れを制御する文を流れ制御文と呼び、for 文や if else 文などがあります。

1.　for 文の書式

for 文は決められた回数の繰り返しに使います。たとえば、

```
        ↓初期値
  for (i=1;i<=10;i++){
            ↑     ↑増減式
            終了条件式
  }
```

は、変数 i を 1 から始め、10 以下の間、i を+1 しながら、{と}で囲まれた範囲（ブロック）を繰り返します。{}内に書く文が 1 つのときは{}を省略できます。繰り返しのことをループと呼びます。for 文の繰り返しを制御している変数のことをループ変数と呼びます。

ループ変数は、

```
var i;
for (i=0;i<=10;i++)
```

のように通常の変数のように宣言することもできますが、for 文の中で宣言することもできます。

```
for (var i=0;i<=10;i++)
```

□for 文の例

for (i=5;i<=10;i++)

・・・i を 5 から始め i を+1 しながら 10 まで繰り返す。6 回繰り返すことになる。

for (i=10;i>0;i--)

・・・i を 10 から始め、i を-1 しながら 0 より大きい間繰り返す。10 回繰り返すことになる。

for (x=-2.0;x<=2.0;x+=0.5)

・・・x を-2.0 から始め,+0.5 しながら 2.0 になるまで繰り返す。

2. 増分演算子、減分演算子

i++は変数 i の内容を+1 します。i--は変数 i の内容を-1 します。++を増分演算子（インクリメント演算子)、--を減分演算子（デクリメント演算子）と呼びます。i++は i=i+1 と同じ意味、i--は i=i-1 と同じ意味です。i+1 の＋は加算演算子、i-1 の-は減算演算子です。

x+=0.5 は x の内容を+0.5 します。x=x+0.5 と同じ意味です。+=を複合代入演算子と呼びます。

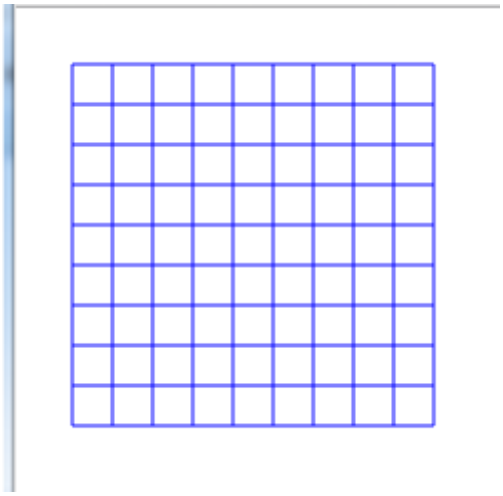
「例題 1-4」始点の x 座標を 20、終点の x 座標を 200 とする直線を y 座標を 20～200 まで 20 きざみに変化させて描きます。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="canvas" width="400" height="400"></canvas>
<script type="text/javascript">
    var canvas = document.getElementById("canvas");
    if(canvas.getContext){
        var context = canvas.getContext("2d");
        context.strokeStyle = "blue"; //  描画色
        for (var y=20;y<=200;y+=20){
            context.beginPath();          // パスの開始
            context.moveTo(20, y);
            context.lineTo(200, y);
            context.stroke();             // パスの描画
        }
    }
</script>
</body>
</html>
```

「練習問題 1-4」始点の y 座標を 20、終点の y 座標を 200 とする直線を x 座標を 20～200 まで 20 きざみに変化させて描く処理を追加しなさい。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="canvas" width="400" height="400"></canvas>
<script type="text/javascript">
    var canvas = document.getElementById("canvas");
    if(canvas.getContext){
        var context = canvas.getContext("2d");
        context.strokeStyle = "blue";
        for (var x=20;x<=200;x+=20){ // 縦線
            context.beginPath();
            ____①____;
            ____②____;
            context.stroke();
        }
        for (var y=20;y<=200;y+=20){ // 横線
            context.beginPath();
            context.moveTo(20, y);
            context.lineTo(200, y);
            context.stroke();
        }
    }
```

```
</script>
</body>
</html>
```



1-5　if else 文

1.　if else 文の書式

条件を判定し、その判定に応じて実行する処理を変えるには if else 文を使います。

```
if (条件式) {
    文 1　←条件を満たしたときに実行される文
}
else {
    文 2　←条件を満たさないときに実行される文
}
```

else 節に書くものがなければ else 節全体を省略します。{}内に書く文が 1 つの時は{}を省略することができます。

条件式としては、次のようなものを書きます。条件式を満たしたときを真、満たさなかったときを偽と呼びます。

a>1	変数 a が 1 より大きいとき真
a==1	変数 a が 1 のとき真
a==1 && b==1	変数 a が 1 でかつ変数 b が 1 のとき真
a==1 || b==1	変数 a が 1 または変数 b が 1 のとき真

2.　比較演算子

条件式において、大きい、小さい、等しいなどの大小比較を行う演算子を比較演算子と呼び次の 6 つがあります。等しいは==と=を 2 つ書くことに注意してください。

比較演算子	意味
>	左辺は右辺より大きい。
>=	左辺は右辺より大きいか等しい。
<	左辺は右辺より小さい。
<=	左辺は右辺より小さいか等しい。
==	左辺と右辺は等しい。
!=	左辺と右辺は等しくない。

3. 論理演算子

1 つの条件式の真と偽を否定（反転）する!演算子、2 つの条件式を組み合わせて真・偽を判定する&&演算子、||演算子を論理演算子と呼びます。

論理演算子	意味
!	否定（NOT）。真なら偽、偽なら真。
&&	かつ（AND）。2 つの条件式の両方が真のとき真。
||	または(OR)。2 つの条件式のどちらかが真のとき真。

4. 真偽の判定

真偽値は真を true、偽を false という定数を使って表します。真偽値を扱う変数 flag は次のように宣言します。

```
var flag=false;
```

変数 flag の真偽値を反転するには以下のようにします。

```
flag=!flag;
```

変数 flag の値が true かを判定するには、

```
if (flag==true)
```

としますが、これは一般に以下のように簡略した記述ができます。

```
if (flag)
```

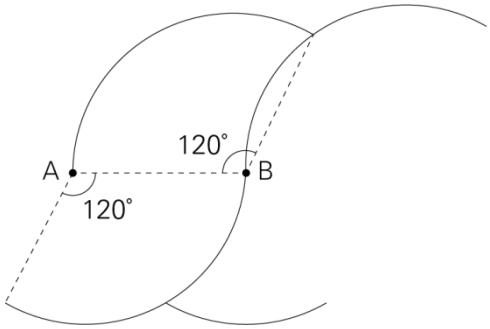
5. ブロックとインデント

for 文や if 文では制御対象となる文が複数になります。このような論理的にひとまとまりの複数の文をブロック（複文）と呼び｛と｝で囲みます。ブロックの中にブロックが入る構造をネスト（入れ子）と呼び、ネストが深くなるたびに、ブロックの書き出す位置を右にずらします。これをインデントと呼びます。インデントの幅は何文字でも良いのですが、通常 4 文字が多いです。以下は<script>ブロックの中に for ブロック、for ブロックの中に if ブロックがネストしています。ネストするたびにインデントが深く（右に書き出し位置が長く）なります。

```
<script type="text/javascript">
    for (・・・){
        if (・・・){

        }
    }
</script>
```

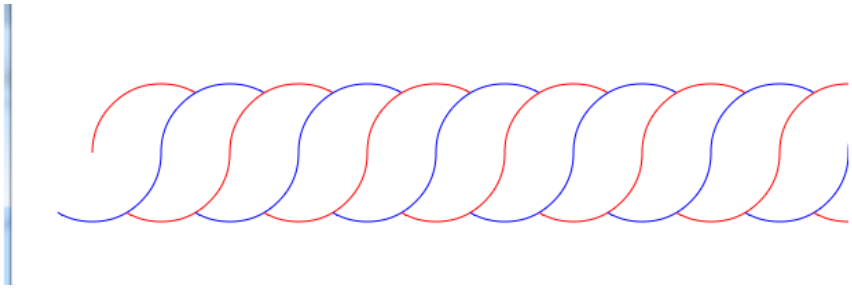
「例題 1-5」A 点を中心に半径 50、角度 120° の円弧を下側に描き、A 点より半径分（50）離れた B 点を中心に角度 120° の円弧を上側に描きます。この操作を繰り返すことでしめ縄模様が描けます。描く円弧の色を変数 flag を使って赤と青の交互に描きます。

図 1-1 しめ縄模様




```
<!DOCTYPE html>
<html>
<body>
<canvas id="canvas" width="600" height="400"></canvas>
<script type="text/javascript">
    var canvas = document.getElementById("canvas");
    if(canvas.getContext){
        var context = canvas.getContext("2d");
        context.strokeStyle = "blue";
        var flag=false;
        for (var x=50;x<=600;x+=50){
            context.beginPath();
            context.arc(x, 100, 50, 0,120*Math.PI/180, false);
            context.stroke();
```

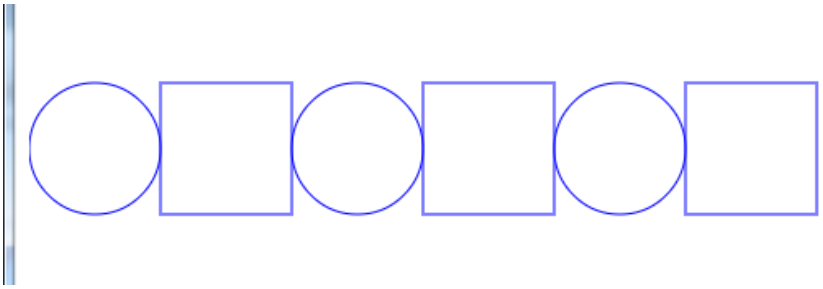
```
                if (flag)
                    context.strokeStyle = "blue"; //  青
                else
                    context.strokeStyle = "red"; //  赤

                context.beginPath();
                context.arc(x+50, 100, 50, 180*Math.PI/180,300*Math.PI/180, false);
                context.stroke();
                flag=!flag;
            }
        }
</script>
</body>
</html>
```



「練習問題 1-5」円と四角を交互に描きなさい。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="canvas" width="600" height="400"></canvas>
<script type="text/javascript">
    var canvas = document.getElementById("canvas");
    if(canvas.getContext){
        var context = canvas.getContext("2d");
        context.strokeStyle = "blue";
        var flag=false;
        for (var x=0;x<=400;x+=80){
```

```
                if (flag){
                    ____________①____________;
                }
                else {
                    context.beginPath();
                    ____________②____________;
                    context.stroke();
                }
                flag=!flag;
            }
        }
</script>
</body>
</html>
```



1-6　二重ループ

ループの中にループが何重にも入る構造を多重ループと呼びます。for 文の二重ループは以下のような構造です。

```
for (var i=1;i<=2;i++){
    for (var j=1;j<=3;j++){
        A
    }
}
```

ループ変数 i が外ループ、ループ変数 j が内ループを管理します。この二重ループは次のように動作します。

外側のループを開始し、i が 1 のまま内側のループの j を 1～3 まで繰り返します。内側ループの繰り返しが終了すると外側のループ変数 i が 2 となり、再び内側のループを繰り返します。

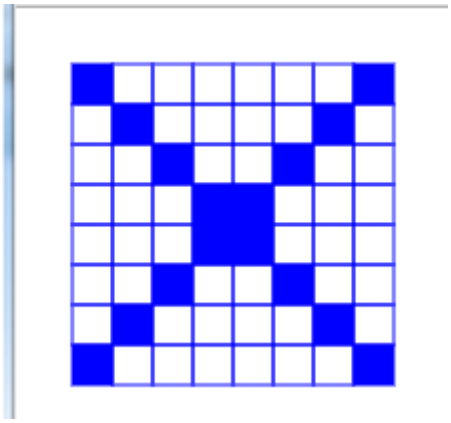
上の二重ループにおいて、A 部におけるループ変数 i と j の値をトレースすると次のようになります。

i	j
1	1
	2
	3
2	1
	2
	3

「例題 1-6」縦 8×横 8 の矩形を青で描きます。縦をループ変数 i、横をループ変数 j で管理したとき、i==j（対角線の要素）のときと i+j==9（逆対角線の要素）のときに矩形内部を青で塗ります。結果として「X」の文字が描けます。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="canvas" width="400" height="400"></canvas>
<script type="text/javascript">
```

```
    var canvas = document.getElementById("canvas");
    if(canvas.getContext){
        var context = canvas.getContext("2d");
        context.strokeStyle = "blue"; //  輪郭の色
        context.fillStyle = "blue"; //  塗る色
        for (var i=1;i<=8;i++){
            for (var j=1;j<=8;j++){
                context.strokeRect(20*j, 20*i, 20, 20);
                if (i==j || i+j==9)
                    context.fillRect(20*j, 20*i, 20, 20);
            }
        }
    }
</script>
</body>
</html>
```



「練習問題 1-6」N の文字を描きなさい。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="canvas" width="400" height="400"></canvas>
<script type="text/javascript">
	var canvas = document.getElementById("canvas");
	if(canvas.getContext){
		var context = canvas.getContext("2d");
		context.strokeStyle = "blue"; //  輪郭の色
		context.fillStyle = "blue"; //  塗る色
		for (var i=1;i<=8;i++){
			for (var j=1;j<=8;j++){
				context.strokeRect(20*j, 20*i, 20, 20);
				if (_____①_____)
					context.fillRect(20*j, 20*i, 20, 20);
			}
		}
	}
</script>
</body>
</html>
```